Byzantine Vector Consensus in Complete Graphs *

Nitin H. Vaidya
University of Illinois at Urbana-Champaign
nhv@illinois.edu
Phone: +1 217-265-5414

Vijay K. Garg
University of Texas at Austin
garg@ece.utexas.edu
Phone: +1 512-471-9424

February 11, 2013

Abstract

Consider a network of n processes each of which has a d-dimensional vector of reals as its *input*. Each process can communicate directly with all the processes in the system; thus the communication network is a *complete graph*. All the communication channels are reliable and FIFO (first-in-first-out). The problem of *Byzantine vector consensus* (BVC) requires agreement on a d-dimensional vector that is in the *convex hull* of the d-dimensional input vectors at the non-faulty processes. We obtain the following results for Byzantine vector consensus in *complete graphs* while tolerating up to f Byzantine failures:

- We prove that in a synchronous system, $n \geq \max(\ 3f+1,\ (d+1)f+1\ )$ is necessary and sufficient for achieving Byzantine vector consensus.
- In an asynchronous system, it is known that *exact* consensus is impossible in presence of faulty processes. For an asynchronous system, we prove that $n \geq (d+2)f+1$ is necessary and sufficient to achieve *approximate* Byzantine vector consensus.

Our sufficiency proofs are constructive. We show sufficiency by providing explicit algorithms that solve exact BVC in synchronous systems, and approximate BVC in asynchronous systems.

We also obtain tight bounds on the number of processes for achieving BVC using algorithms that are restricted to a simpler communication pattern.

*This research is supported in part by National Science Foundation awards CNS-1059540 and CNS-1115808 and the Cullen Trust for Higher Education. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies or the U.S. government.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
11 FEB 2013		**00-00-2013 to 00-00-2013**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Byzantine Vector Consensus in Complete Graphs	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Illinois at Urbana-Champaign,Department of Electrical and Computer Engineering,Urbana,IL,61801**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Consider a network of n processes each of which has a d-dimensional vector of reals as its input. Each process can communicate directly with all the processes in the system thus the communication network is a complete graph. All the communication channels are reliable and FIFO (rst-in- rst-out). The problem of Byzantine vector consensus (BVC) requires agreement on a d-dimensional vector that is in the convex hull of the d-dimensional input vectors at the non-faulty processes. We obtain the following results for Byzantine vector consensus in complete graphs while tolerating up to f Byzantine failures We prove that in a synchronous system, n max(3f+1; (d+1)f+1) is necessary and su cient for achieving Byzantine vector consensus. In an asynchronous system, it is known that exact consensus is impossible in pres- ence of faulty processes. For an asynchronous system, we prove that n (d+2)f+1 is necessary and su cient to achieve approximate Byzantine vector consensus. Our su ciency proofs are constructive. We show su ciency by providing explicit algo- rithms that solve exact BVC in synchronous systems, and approximate BVC in asyn- chronous systems. We also obtain tight bounds on the number of processes for achieving BVC using algorithms that are restricted to a simpler communication pattern.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**20**	

1 Introduction

This paper addresses *Byzantine vector consensus* (BVC), wherein the input at each process is a d-dimensional vector of reals, and each process is expected to decide on a *decision vector* that is in the *convex hull* of the input vectors at the non-faulty processes. The system consists of n processes in $\mathcal{P} = \{p_1, p_2, \cdots, p_n\}$. We assume $n > 1$, since consensus is trivial for $n = 1$. At most f processes may be Byzantine faulty, and may behave arbitrarily [12]. All processes can communicate with each other directly on *reliable FIFO* (first-in first-out) channels. Thus, the communication network is a *complete graph*. The input *vector* at each process may also be viewed as a *point* in the d-dimensional Euclidean space $\mathbf{R}^d$, where $d > 0$ is a finite integer. Due to this correspondence, we use the terms *point* and *vector* interchangeably. Similarly, we interchangeably refer to the d *elements* of a vector as *coordinates*. We consider two versions of the Byzantine vector consensus (BVC) problem, *Exact BVC* and *Approximate BVC*.

Exact BVC: Exact Byzantine vector consensus must satisfy the following three conditions.

- *Agreement*: The decision (or output) vector at all the non-faulty processes must be identical.
- *Validity*: The decision vector at each non-faulty process must be in the convex hull of the input vectors at the non-faulty processes.
- *Termination*: Each non-faulty process must terminate after a finite amount of time.

The traditional consensus problem [13, 10] is obtained when $d = 1$; we refer to this as *scalar* consensus. $n \geq 3f + 1$ is known to be necessary and sufficient for achieving Byzantine *scalar* consensus in complete graphs [12, 13]. We observe that simply performing *scalar* consensus on each dimension of the input vectors independently does not solve the *vector* consensus problem. In particular, even if validity condition for *scalar consensus* is satisfied for each dimension of the vector separately, the above *validity* condition of vector consensus may not necessarily be satisfied. For instance, suppose that there are four processes, with one faulty process. Processes p_1, p_2 and p_3 are non-faulty, and have the following 3-dimensional input vectors, respectively: $\mathbf{x}_1 = [\frac{2}{3}, \frac{1}{6}, \frac{1}{6}]$, $\mathbf{x}_2 = [\frac{1}{6}, \frac{2}{3}, \frac{1}{6}]$, $\mathbf{x}_3 = [\frac{1}{6}, \frac{1}{6}, \frac{2}{3}]$. Process p_4 is faulty. If we perform Byzantine *scalar* consensus on each dimension of the vector separately, then the processes may possibly agree on the decision vector $[\frac{1}{6}, \frac{1}{6}, \frac{1}{6}]$, each element of which satisfies *scalar* validity condition *along each dimension* separately; however, this decision vector *does not* satisfy the validity condition for BVC because it is *not* in the convex hull of input vectors of non-faulty processes. In this example, since every non-faulty process has a probability vector as its input vector, BVC validity condition requires that the decision vector should also be a probability vector. In general, for many optimization problems [4], the set of feasible solutions is a convex set in Euclidean space. Assuming that every non-faulty process proposes a feasible solution, BVC guarantees that the vector decided is also a feasible solution. Using scalar consensus along each dimension is not sufficient to provide this guarantee.

Approximate BVC: In an *asynchronous* system, processes may take steps at arbitrary relative speeds, and there is no fixed upper bound on message delays. Fischer, Lynch and Paterson [9] proved that exact consensus is impossible in asynchronous systems in the presence of even a single crash failure. As a way to circumvent this impossibility result, Dolev et al. [5] introduced the notion of *approximate* consensus, and proved the correctness of an algorithm for approximate Byzantine *scalar* consensus in asynchronous systems when $n \geq 5f + 1$. Subsequently, Abraham, Amit and

Dolev [1] established that approximate Byzantine *scalar* consensus is possible in asynchronous systems if $n \geq 3f + 1$. Other algorithms for approximate consensus have also been proposed (e.g., [3, 8]). We extend the notion of approximate consensus to *vector* consensus. *Approximate BVC* must satisfy the following conditions:

- ϵ-*Agreement*: For $1 \leq l \leq d$, the l-th elements of the decision vectors at any two non-faulty processes must be within ϵ of each other, where $\epsilon > 0$ is a pre-defined constant.
- *Validity*: The decision vector at each non-faulty process must be in the convex hull of the input vectors at the non-faulty processes.
- *Termination*: Each non-faulty process must terminate after a finite amount of time.

The main contribution of this paper is to establish the following bounds for *complete graphs*.

- In a synchronous system, $n \geq \max(3f + 1, (d + 1)f + 1)$ is necessary and sufficient for *Exact BVC* in presence of up to f Byzantine faulty processes. (Theorems 1 and 3).
- In an asynchronous system, $n \geq (d+2)f+1$ is necessary and sufficient for *Approximate BVC* in presence of up to f Byzantine faulty processes. (Theorems 4 and 5).

Observe that the two bounds above are different when $d > 1$, unlike the case of $d = 1$ (i.e., scalar consensus). When $d = 1$, in a complete graph, $3f + 1$ processes are sufficient for exact consensus in synchronous systems, as well as approximate consensus in asynchronous systems [1]. For $d > 1$, the lower bound for asynchronous systems is larger by f compared to the bound for synchronous systems.

In prior literature, the term *vector consensus* has also been used to refer to another form of consensus, wherein the input at each process is a *scalar*, but the agreement is on a vector containing these scalars [7, 16]. Thus, our results are for a different notion of consensus.

Simpler (Restricted) Algorithm Structure

In prior literature, iterative algorithms with very simple structure have been proposed to achieve *approximate* consensus, including asynchronous approximate Byzantine scalar consensus [5] in *complete* graphs, and synchronous as well as asynchronous approximate Byzantine consensus in *incomplete* graphs [18]. Section 4 extends these simple structures to vector consensus in complete graphs, and obtains the following tight bounds: (i) $n \geq (d + 2)f + 1$ for synchronous systems, and (ii) $n \geq (d + 4)f + 1$ for asynchronous systems. Observe that the bound for the simple iterative algorithms in asynchronous systems is larger by $2f$ when compared to the bound stated earlier: this is the cost of restricting the algorithm structure. This $2f$ gap is analogous to that between the sufficient condition of $n \geq 3f+1$ for asynchronous scalar consensus proved by Abraham, Amit and Dolev [1], the sufficient condition of $n \geq 5f + 1$ demonstrated by Dolev et al. [5] using a simpler algorithm.

Our Notations

Many notations introduced throughout the paper are also summarized in Appendix A. We use operator $|\,.\,|$ to obtain the size of a *multiset* or a *set*. We use operator $\|\ .\ \|$ to obtain the absolute value of a scalar.

2 Synchronous Systems

In this section, we derive necessary and sufficient conditions for exact BVC in a synchronous system with up to f faulty processes. The discussion in the rest of this paper assumes that the network is a *complete graph*, even if this is not stated explicitly in all the results.

2.1 Necessary Condition for Exact BVC

Theorem 1 $n \geq \max(3f+1, (d+1)f+1)$ *is necessary for Exact BVC in a synchronous system.*

Proof: From [12, 13], we know that, for $d = 1$ (i.e., scalar consensus), $n \geq 3f+1$ is a necessary condition for achieving exact Byzantine consensus in presence of up to f faults. If we were to restrict the d-dimensional input vectors to have identical d elements, then the problem of vector consensus reduces to scalar consensus. Therefore, $n \geq 3f+1$ is also a necessary condition for *Exact BVC* for arbitrary d. Now we prove that $n \geq (d+1)f+1$ is also a necessary condition.

First consider the case when $f = 1$, i.e., at most one process may be faulty. Since none of the non-faulty processes know which process, if any, is faulty, as elaborated in Appendix C, the decision vector must be in the convex hull of each multiset containing the input vectors of $n-1$ of the processes (there are n such multisets).[1] Thus, this intersection must be non-empty, for all possible input vectors at the n processes. (Appendix C provides further clarification.) We now show that the intersection may be empty when $n = d+1$; thus, $n = d+1$ is not sufficient for $f = 1$.

Suppose that $n = d+1$. Consider the following set of input vectors. The input vector of process p_i, where $1 \leq i \leq d$, is a vector whose i-th element is 1, and the remaining elements are 0. The input vector at process p_{d+1} is the all-0 vector (i.e., the vector with all elements 0). Note that the d input vectors at $p_1, \cdots, p_d$ form the standard basis for the d-dimensional vector space. Also, none of the $d+1$ input vectors can be represented as a convex combination of the remaining d input vectors. For $1 \leq i \leq d+1$, let Q_i denote the convex hull of the inputs at the $n-1 = d$ processes in $\mathcal{P} - \{p_i\}$. We now argue that $\cap_{i=1}^{d+1} Q_i$ is empty.

For $1 \leq i \leq d$, observe that for all the points in Q_i, the i-th coordinate is 0. Thus, any point that belongs to the intersection $\cap_{i=1}^{d} Q_i$ must have all its coordinates 0. That is, only the all-0 vector belongs to $\cap_{i=1}^{d} Q_i$. Now consider Q_{d+1}, which is the convex hull of the inputs at the first d processes. Due to the choice of the inputs at the first d processes, the origin (i.e., the all-0 vector) does not belong to Q_{d+1}. From the earlier observation on $\cap_{i=1}^{d} Q_i$, it then follows that $\cap_{i=1}^{d+1} Q_i = \emptyset$. Therefore, the *Exact BVC* problem for $f = 1$ cannot be solved with $n = d+1$. Thus, $n = d+1$ is not sufficient. It should be easy to see that $n \leq d+1$ is also not sufficient. Thus, $n \geq d+2$ is a necessary condition for $f = 1$.

Now consider the case of $f > 1$. Using the commonly used simulation approach [12], we can prove that $(d+1)f$ processes are not sufficient. In this approach, f *simulated processes* are implemented by a single process. If a correct algorithm were to exist for tolerating f faults among $(d+1)f$ processes, then we can obtain a correct algorithm to tolerate a single failure among $d+1$ processes, contradicting our result above. Thus, $n \geq (d+1)f+1$ is necessary for $f \geq 1$. (For $f = 0$, the necessary condition holds trivially.) □

[1] Since the state of two processes may be identical, we use a *multiset* to represent the collection of the states of a subset of processes. Appendix B elaborates on the notion of multisets.

2.2 Sufficient Condition for Exact BVC

We now present an algorithm for Exact BVC in a synchronous system, and prove its correctness in a complete graph with $n \geq \max(3f+1, (d+1)f+1)$. The algorithm uses function $\Gamma(Y)$ defined below, where Y is a multiset of points. $\mathcal{H}(T)$ denotes the convex hull of a multiset T.

$$\Gamma(Y) = \cap_{T \subseteq Y, |T|=|Y|-f} \mathcal{H}(T). \tag{1}$$

The intersection above is over the convex hulls of all subsets of Y of size $|Y| - f$.

Exact BVC algorithm for $n \geq \max(3f+1, (d+1)f+1)$:

---

1. Each process uses a scalar *Byzantine broadcast* algorithm (such as [12, 6]) to broadcast each element of its input vector to all the other processes (each element is a scalar). The *Byzantine broadcast* algorithm allows a designated sender to broadcast a scalar value to the other processes, while satisfying the following properties when $n \geq 3f+1$: (i) all the non-faulty processes decide on an identical scalar value, and (ii) if the sender is non-faulty, then the value decided by the non-faulty processes is the sender's proposed (scalar) value. Thus, non-faulty processes can agree on the d elements of the input vector at each of the n processes.

   At the end of the this step, each non-faulty process would have received an *identical* multiset S containing n vectors, such that the vector corresponding to each non-faulty process is identical to the input vector at that process.

2. Each process chooses as its *decision* vector a point in $\Gamma(S)$; all non-faulty processes choose the point identically using a deterministic function. We will soon show that $\Gamma(S)$ is non-empty.

We now prove that the above algorithm is correct. Later, we show how the *decision vector* can be found in Step 2 using linear programming. The proof of correctness of the above algorithm uses the following celebrated theorem by Tverberg [17]:

Theorem 2 (Tverberg's Theorem [17]) *For any integer $f \geq 1$, and for every multiset Y containing at least $(d+1)f+1$ points in $\mathbf{R}^d$, there exists a partition $Y_1, \cdots, Y_{f+1}$ of Y into $f+1$ non-empty multisets such that $\cap_{l=1}^{f+1} \mathcal{H}(Y_l) \neq \emptyset$.*

The points in multiset Y above are not necessarily distinct [17]; thus, the same point may occur multiple times in Y. (Appendix B elaborates on the notion of multisets, and multiset partition.) The partition in Theorem 2 is called a *Tverberg partition*, and the points in $\cap_{l=1}^{f+1} \mathcal{H}(Y_l)$ in Theorem 2 are called *Tverberg points*. Appendix D provides an illustration of a Tverberg partition for points in 2-dimensional space.

The lemma below is used to prove the correctness of the above algorithm, as well as the algorithm presented later in Section 3.

Lemma 1 *For any multiset Y containing at least $(d+1)f+1$ points in $\mathbf{R}^d$, $\Gamma(Y) \neq \emptyset$.*

Proof: Consider a Tverberg partition of Y into $f+1$ non-empty subsets $Y_1, \cdots, Y_{f+1}$, such that the set of Tverberg points $\cap_{l=1}^{f+1} \mathcal{H}(Y_l) \neq \emptyset$. Since $|Y| \geq (d+1)f+1$, by Theorem 2, such a partition exists. By (1) we have

$$\Gamma(Y) = \cap_{T \subseteq Y, |T|=|Y|-f} \mathcal{H}(T). \tag{2}$$

Consider any T in (2). Since $|T| = |Y| - f$ and there are $f+1$ subsets in the Tverberg partition of Y, T excludes elements from at most f of these subsets. Thus, T contains at least one subset from the partition. Therefore, for **each** T, $\cap_{l=1}^{f+1} \mathcal{H}(Y_l) \subseteq \mathcal{H}(T)$. Hence, from (2), it follows that $\cap_{l=1}^{f+1} \mathcal{H}(Y_l) \subseteq \Gamma(Y)$. Also, because $\cap_{l=1}^{f+1} \mathcal{H}(Y_l) \neq \emptyset$, it now follows that $\Gamma(Y) \neq \emptyset$. □

We can now prove the correctness of our Exact BVC algorithm.

Theorem 3 *$n \geq \max(3f+1, (d+1)f+1)$ is sufficient for achieving Exact BVC in a synchronous system.*

Proof: We prove that the above *Exact BVC* algorithm is correct when $n \geq \max(3f+1, (d+1)f+1)$. The *termination* condition holds because the *Byzantine broadcast* algorithm used in Step 1 terminates in finite time. Since $|S| = n \geq (d+1)f+1$, by Lemma 1, $\Gamma(S) \neq \emptyset$. By (1) we have

$$\Gamma(S) = \cap_{T \subseteq S, |T| = |S| - f} \mathcal{H}(T). \tag{3}$$

At least one of the multisets T in (3), say T^*, must contain the inputs of *only* non-faulty processes, because $|T| = |S| - f = n - f$, and there are at most f faulty processes. By definition of $\Gamma(S)$, $\Gamma(S) \subseteq \mathcal{H}(T^*)$. Then, from the definition of T^*, and the fact that the decision vector is chosen from $\Gamma(S)$, the *validity* condition follows.

Agreement condition holds because all the non-faulty processes have identical S, and pick as their decision vector a point in $\Gamma(S)$ using a deterministic function in Step 2. □

We now show how Step 2 of the Exact BVC algorithm can be implemented using linear programming. The input to the linear program is $S = \{\mathbf{s}_i \ : \ 1 \leq i \leq n\}$, a multiset of d-dimensional vectors. Our goal is to find a vector $\mathbf{z} \in \Gamma(S)$; or equivalently, find a vector $\mathbf{z}$ that can be expressed as a convex combination of vectors in T for all choices $T \subseteq S$ such that $|T| = n - f$. The linear program uses the following $d + \binom{n}{n-f}(n-f)$ variables.

- $\mathbf{z}_1, ..\mathbf{z}_d$: variables for d elements of vector $\mathbf{z}$.
- $\alpha_{T,i}$: coefficients such that $\mathbf{z}$ can be written as convex combination of vectors in T. We include here only those $n-f$ indices i for which $\mathbf{s}_i \in T$.

For every T, the linear constraints are as follows.

- $\mathbf{z} = \sum_{\mathbf{s}_i \in T} \alpha_{T,i} \, \mathbf{s}_i$ ($\mathbf{z}$ is a linear combination of $\mathbf{s}_i \in T$)
- $\sum_{\mathbf{s}_i \in T} \alpha_{T,i} = 1$ (The sum of all coefficients for a particular T is 1)
- $\alpha_{T,i} \geq 0$ for all $\mathbf{s}_i \in T$.

For every T, we get $d+1+n-f$ linear constraints, yielding a total of $\binom{n}{n-f}(d+1+n-f)$ constraints in $d + \binom{n}{n-f}(n-f)$ variables. Hence, for any *fixed* f, a point in $\Gamma(S)$ can be found in polynomial time by solving a linear program with the number of variables and constraints that are polynomial in n and d (but not in f). However, when f grows with n, the computational complexity is high.

We note here that the above Exact BVC algorithm remains correct if the non-faulty processes identically choose *any point* in $\Gamma(S)$ as the decision vector. In particular, as seen in the proof of Lemma 1, all the Tverberg points are contained in $\Gamma(S)$, therefore, one of the Tverberg points for multiset S may be chosen as the decision vector. It turns out that, for arbitrary d, currently there is no known algorithm with polynomial complexity to compute a Tverberg point for a given multiset [2, 14, 15]. However, in some restricted cases, efficient algorithms are known (e.g., [11]).

3 Asynchronous Systems

We develop a tight necessary and sufficient condition for *approximate* asynchronous BVC.

3.1 Necessary Condition for Approximate Asynchronous BVC

Theorem 4 *$n \geq (d+2)f+1$ is necessary for approximate BVC in an asynchronous system.*

Proof: We first consider the case of $f = 1$. Suppose that a correct algorithm exists for $n = d+2$. Denote by $\mathbf{x}_k$ the input vector at each process p_k. Now consider a process p_i, where $1 \leq i \leq d+1$. Since a correct algorithm must tolerate one failure, process p_i must terminate in all executions in which process p_{d+2} does not take any steps. Suppose that all the processes are non-faulty, but process p_{d+2} does not take any steps until all the other processes terminate. At the time when process p_i terminates ($1 \leq i \leq d+1$), it cannot distinguish between the following $d+1$ scenarios:

- Process p_{d+2} has crashed: In this case, to satisfy the *validity* condition, the decision of process p_i must be in the convex hull of the inputs of processes $p_1, p_2, \cdots, p_{d+1}$. That is, the decision vector must be in the convex hull of X_i^{d+2} defined below.

$$X_i^{d+2} \quad = \quad \{\mathbf{x}_k \ : \ 1 \leq k \leq d+1\} \tag{4}$$

  $\mathbf{x}_{d+2}$ is not included above, because until process p_i terminates, p_{d+2} does not take any steps (so p_i cannot learn any information about $\mathbf{x}_{d+2}$).

- Process p_j ($j \neq i$, $1 \leq j \leq d+1$) is faulty, and process p_{d+2} is slow, and hence p_{d+2} has not taken any steps yet: Recall that we are considering p_i at the time when it terminates. Since process p_{d+2} has not taken any steps yet, process p_i cannot have any information about the input at p_{d+2}. Also, in this scenario p_j may be faulty, therefore, process p_i cannot trust the correctness of the input at p_j. Thus, to satisfy the validity condition, the decision of process p_i must be in the convex hull of X_i^j defined below.

$$X_i^j \quad = \quad \{\mathbf{x}_k \ : \ k \neq j \text{ and } 1 \leq k \leq d+1\} \tag{5}$$

The decision vector of process p_i must be valid independent of which of the above $d+1$ scenarios actually occurred. Therefore, observing that $\mathcal{H}(X_i^{d+2}) \supseteq \mathcal{H}(X_i^j)$, where $j \neq i$, we conclude that the decision vector must be in

$$\cap_{j \neq i, 1 \leq j \leq d+1} \ \mathcal{H}(X_i^j) \tag{6}$$

Recall that $\epsilon > 0$ is the parameter of the ϵ-agreement condition in Section 1. For $1 \leq i \leq d$, suppose that the i-th element of input vector $\mathbf{x}_i$ is 4ϵ, and the remaining $d-1$ elements are 0. Also suppose that $\mathbf{x}_{d+1}$ and $\mathbf{x}_{d+2}$ are both equal to the all-0 vector.

Let us consider process p_{d+1}. In this case, $\mathcal{H}(X_{d+1}^j)$ for $j \leq d$ only contains vectors whose j-th element is 0. Thus, the intersection of all the convex hulls in (6) only contains the all-0 vector, which, in fact, equals $\mathbf{x}_{d+1}$. Thus, the decision vector of process p_{d+1} must be equal to $\mathbf{x}_{d+1}$. We can similarly show that for each p_i, $1 \leq i \leq d+1$, the intersection in (6) only contains vector $\mathbf{x}_i$, and therefore, the decision vector of process p_i must be equal to its input $\mathbf{x}_i$. The input vectors at each pair of processes in $p_1, \cdots, p_{d+1}$ differ by 4ϵ in at least one element. This implies that the

ϵ-agreement condition is not satisfied. Therefore, $n = d+2$ is not sufficient for $f = 1$. It should be easy to see that $n \leq d+2$ is also not sufficient.

For the case when $f > 1$, by using a *simulation* similar to the proof of Theorem 1, we can now show that $n \leq (d+2)f$ is not sufficient. Thus, $n \geq (d+2)f+1$ is necessary for $f \geq 1$. (For $f = 0$, the necessary condition holds trivially.) □

3.2 Sufficient Condition for Approximate Asynchronous BVC

We will prove that $n \geq (d+2)f+1$ is sufficient by proving the correctness of an algorithm presented in this section. The proposed algorithm executes in asynchronous rounds. Each process p_i maintains a local state $\mathbf{v}_i$, which is a d-dimensional vector. We will refer to the value of $\mathbf{v}_i$ at the *end* of the t-th round performed by process p_i as $\mathbf{v}_i[t]$. Thus, $\mathbf{v}_i[t-1]$ is the value of $\mathbf{v}_i$ at the *start* of the t-th round of process p_i. The initial value of $\mathbf{v}_i$, namely $\mathbf{v}_i[0]$, is equal to p_i's *input* vector, denoted as $\mathbf{x}_i$. The messages sent by each process anytime during its t-th round are tagged by the round number t. This allows a process p_i in its round t to determine, despite the asynchrony, whether a message received from another process p_j was sent by p_j in p_j's round t.

The proposed algorithm is obtained by suitably modifying a *scalar* consensus algorithm presented by Abraham, Amit and Dolev [1] to achieve asynchronous approximate Byzantine scalar consensus among $3f+1$ processes. We will refer to the algorithm in [1] as the AAD algorithm. We first present a brief overview of the AAD algorithm, and describe its properties. We adopt our notation above when describing the AAD algorithm (the notation differs from [1]). One key difference is that, in our proposed algorithm $\mathbf{v}_i[t]$ is a vector, whereas in AAD description below, it is considered a scalar. The AAD algorithm may be viewed as consisting of three components:

1. *AAD component #1:* In each round t, the AAD algorithm requires each process to communicate its state $\mathbf{v}_i[t-1]$ to other processes using a mechanism that achieves the properties described next. AAD ensures that each non-faulty process p_i in its round t obtains a set $B_i[t]$ containing at least $n-f$ tuples of the form $(p_j, \mathbf{w}_j, t)$, such that the following properties hold:

   - (Property 1) For any two non-faulty processes p_i and p_j:

     $$|B_i[t] \cap B_j[t]| \geq n-f \tag{7}$$

     That is, p_i and p_j learn at least $n-f$ identical tuples.
   - (Property 2) If $(p_l, \mathbf{w}_l, t)$ and $(p_k, \mathbf{w}_k, t)$ are both in $B_i[t]$, then $p_l \neq p_k$. That is, $B_i[t]$ contains at most one tuple for each process.
   - (Property 3) If p_k is non-faulty, and $(p_k, \mathbf{w}_k, t) \in B_i[t]$, then $\mathbf{w}_k = \mathbf{v}_k[t-1]$. That is, for any non-faulty process p_k, $B_i[t]$ may only contain the tuple $(p_k, \mathbf{v}_k[t-1], t)$. (However, it is possible that, corresponding to some non-faulty process, $B_i[t]$ does not contain a tuple at all.)

2. *AAD component #2:* Process p_i, having obtained set $B_i[t]$ above, computes its new state $\mathbf{v}_i[t]$ as a function of the tuples in $B_i[t]$. The primary difference between our proposed algorithm and AAD is in this step. The computation of $\mathbf{v}_i[t]$ in AAD is designed to be correct for scalar inputs (and scalar decision), whereas our approach applies to d-dimensional vectors.

3. *AAD component #3:* AAD also includes a sub-algorithm that allows the non-faulty processes to determine when to terminate their computation. Initially, the processes cooperate to

estimate a quantity δ as a function of the input values at various processes. Different non-faulty processes may estimate different values for δ, since the estimate is affected by the behavior of faulty processes and message delays. Each process then uses $1 + \lceil \log_2 \frac{\delta}{\epsilon} \rceil$ as the threshold on the minimum number of rounds necessary for the non-faulty processes to converge within ϵ of each other. The base of the logarithm above is 2, because the range of the values at the non-faulty processes is shown to shrink by a factor of $\frac{1}{2}$ after each asynchronous round of AAD [1]. Subsequently, when the processes reach respective thresholds on the rounds, they exchange additional messages. After an adequate number of processes announce that they have reached their threshold, all the non-faulty processes may terminate.

It turns out that the Properties 1, 2 and 3 hold even if *Component #1* of AAD is used with $\mathbf{v}_i[t]$ as a *vector*. We exploit these properties in our algorithm below. The proposed algorithm below uses a function Φ, which takes a set, say set B, containing tuples of the form $(p_k, \mathbf{w}_k, t)$, and returns a multiset containing the points (i.e., $\mathbf{w}_k$). Formally,

$$\Phi(B) \quad = \quad \{\mathbf{w}_k \ : \ (p_k, \mathbf{w}_k, t) \in B\} \tag{8}$$

A mechanism similar to that in AAD may potentially be used to achieve termination for the approximate BVC algorithm below as well. The main difference from AAD would be in the manner in which the threshold on the number of rounds necessary is computed. However, for brevity, we simplify our algorithm by assuming that there exists an upper bound U and a lower bound ν on the values of the d elements in the inputs vectors at non-faulty processes, and that these bounds are known *a priori*. Thus, all the elements in each input vector will be $\leq U$ and $\geq \nu$. This assumption holds in many practical systems, because the input vector elements represent quantities that are constrained. For instance, if the input vectors are probability vectors, then $U = 1$ and $\nu = 0$. If the input vectors represent locations in 3-dimensional space occupied by mobile robots, then U and ν are determined by the boundary of the region in which the robots are allowed to operate. The advantage of the AAD-like solution over our simple approach is that, depending on the actual inputs, the algorithm may potentially terminate sooner, and the AAD mechanism prevents faulty processes from causing the non-faulty processes to run longer than necessary. However, the simple static approach for termination presently suffices to prove the correctness of our approximate BVC algorithm, as shown later.

Asynchronous Approximate BVC algorithm for $n \geq (d+2)f + 1$:

---

1. In the t-th round, each non-faulty process uses the mechanism in *Component #1* of the AAD algorithm to obtain a set $B_i[t]$ containing at least $n - f$ tuples, such that $B_i[t]$ satisfies properties 1, 2, and 3 described earlier for AAD. While these properties were proved in [1] for scalar states, the correctness of the properties also holds when $\mathbf{v}_i$ is a vector.

2. In the t-th round, after obtaining set $B_i[t]$, process p_i computes its new state $\mathbf{v}_i[t]$ as follows. Form a multiset Z_i using the steps below:

   - Initialize Z_i as empty.
   - For each $C \subseteq B_i[t]$ such that $|C| = n - f \geq (d+1)f + 1$, add to Z_i one deterministically chosen point from $\Gamma(\Phi(C))$. Since $|\Phi(C)| = |C| \geq (d+1)f + 1$, by Lemma 1, $\Gamma(\Phi(C))$ is non-empty.

Note that $|Z_i| = \binom{|B_i[t]|}{n-f} \leq \binom{n}{n-f}$. Calculate

$$\mathbf{v}_i[t] = \frac{\sum_{\mathbf{z} \in Z_i} \mathbf{z}}{|Z_i|} \tag{9}$$

3. Each non-faulty process terminates after $1 + \lceil \log_{1/(1-\gamma)} \frac{U-\nu}{\epsilon} \rceil$ rounds, where γ ($0 < \gamma < 1$) is a constant defined later in (11). Recall that ϵ is the parameter of the ϵ-agreement condition.

In Step 2 above, we consider $\binom{|B_i[t]|}{n-f}$ subsets C of $B_i[t]$, each subset being of size $n-f$. As elaborated in Appendix F, it is possible to reduce the number of subsets explored to just $n-f$. This optimization will reduce the computational complexity of Step 2, but it is not necessary for correctness of the algorithm.

Theorem 5 *$n \geq (d+2)f + 1$ is sufficient for approximate BVC in an asynchronous system.*

Proof: Without loss of generality, suppose that m processes $p_1, p_2, \cdots p_m$ are non-faulty, where $m \geq n - f$, and the remaining $n - m$ processes are faulty. In the proof, we will often omit the round index $[t]$ in $B_i[t]$, since the index should be clear from the context. In this proof, we consider the steps taken by the non-faulty processes in their respective t-th rounds, where $t > 0$. We now define a *valid* point. The definition is used later in the proof.

Definition 1 *A point* $\mathbf{r}$ *is said to be* valid *if there exists a representation of* $\mathbf{r}$ *as a convex combination of* $\mathbf{v}_k[t-1]$, $1 \leq k \leq m$. *That is, there exist constants* β_k, *such that* $0 \leq \beta_k \leq 1$ *and* $\sum_{1 \leq k \leq m} \beta_k = 1$, *and*

$$\mathbf{r} = \sum_{1 \leq k \leq m} \beta_k \, \mathbf{v}_k[t-1] \tag{10}$$

β_k is said to be the **weight** *of $\mathbf{v}_k[t-1]$ in the above convex combination.*

In general, there may exist multiple such convex combination representations of a *valid* point $\mathbf{r}$. Observe that at least one of the weights in any such convex combination must be $\geq \frac{1}{m} \geq \frac{1}{n}$.

For the convenience of the readers, we break up the rest of this proof into three parts.

Part I: At a non-faulty process p_i, consider any $C \subseteq B_i$ such that $|C| = n - f$ (as in Step 2 of the algorithm). Since $|\Phi(C)| = |C| = n - f \geq (d+1)f + 1$, by Lemma 1, $\Gamma(\Phi(C)) \neq \emptyset$. So Z_i will contain a point from $\Gamma(\Phi(C))$ for each C.

Now, $C \subseteq B_i$, $|\Phi(C)| = n - f$, and there are at most f faulty processes. Then Property 3 of B_i implies that at least one $(n-2f)$-size subset of $\Phi(C)$ must also be a subset of $\{\mathbf{v}_1[t-1], \mathbf{v}_2[t-1], \cdots, \mathbf{v}_m[t-1]\}$, i.e., contain only the state of non-faulty processes. Therefore, all the points in $\Gamma(\Phi(C))$ must be *valid* (due to (1) and Definition 1). This observation is true for each set C enumerated in Step 2. Therefore, all the points in Z_i computed in Step 2 must be valid. (Recall that we assume processes $p_1, \cdots, p_m$ are non-faulty.)

Part II: Consider any two non-faulty processes p_i and p_j.

- *Observation 1:* As argued in Part I, all the points in Z_i are valid. Therefore, all the points in Z_i can be expressed as convex combinations of the state of non-faulty processes, i.e., $\{\mathbf{v}_1[t-1], \cdots, \mathbf{v}_m[t-1]\}$. Similar observation holds for all the points in Z_j too.

- *Observation 2:* By Property 1 of B_i and B_j,[2]

$$|B_i \cap B_j| \geq n - f.$$

  Therefore, there exists a set $C_{ij} \subseteq B_i \cap B_j$ such that $|C_{ij}| = n - f$. Therefore, Z_i and Z_j both contain one identical point from $\Gamma(\Phi(C_{ij}))$. Suppose that this point is named $\mathbf{z}_{ij}$. As shown in Part I above, $\mathbf{z}_{ij}$ must be *valid*. Therefore, there exists a convex combination representation of $\mathbf{z}_{ij}$ in terms of the states $\{\mathbf{v}_1[t-1], \mathbf{v}_2[t-1], \cdots, \mathbf{v}_m[t-1]\}$ of non-faulty processes. Choose any one such convex combination. There must exist a non-faulty process, say $p_{g(i,j)}$, such that the weight associated with $\mathbf{v}_{g(i,j)}[t-1]$ in the convex combination for $\mathbf{z}_{ij}$ is $\geq \frac{1}{m} \geq \frac{1}{n}$. We can now make the next observation.[3]

- *Observation 3:* Recall from (9) that $\mathbf{v}_i[t]$ is computed as the average of the points in Z_i, and $|Z_i| = \binom{|B_i|}{n-f} \leq \binom{n}{n-f}$. By *Observations 1*, all the points in Z_i are valid, and by *Observation 2*, $\mathbf{z}_{ij} \in Z_i$. These observations together imply that $\mathbf{v}_i[t]$ is also valid, and *there exists* a representation of $\mathbf{v}_i[t]$ as a convex combination of $\{\mathbf{v}_1[t-1], \cdots, \mathbf{v}_m[t-1]\}$, wherein the weight of $\mathbf{v}_{g(i,j)}[t-1]$ is $\geq \frac{1}{n\binom{|B_i|}{n-f}} \geq \frac{1}{n\binom{n}{n-f}}$. Similarly, we can show that *there exists* a representation of $\mathbf{v}_j[t]$ as a convex combination of $\{\mathbf{v}_1[t-1], \cdots, \mathbf{v}_m[t-1]\}$, wherein the weight of $\mathbf{v}_{g(i,j)}[t-1]$ is $\geq \frac{1}{n\binom{n}{n-f}}$. Define

$$\gamma = \frac{1}{n\binom{n}{n-f}} \tag{11}$$

  Consensus is trivial for $n = 1$, so we consider finite $n > 1$. Therefore, $0 < \gamma < 1$.

Part III: Observation 3 above implies that for any $\tau > 0$, $\mathbf{v}_i[\tau]$ is a convex combination of $\{\mathbf{v}_1[\tau-1], \cdots, \mathbf{v}_m[\tau-1]\}$. Applying this observation for $\tau = 1, 2, \cdots, t$, we can conclude that $\mathbf{v}_i[t]$ is a convex combination of $\{\mathbf{v}_1[0], \cdots, \mathbf{v}_m[0]\}$, implying that the proposed algorithm satisfies the **validity** condition for approximate consensus. (Recall that $\mathbf{v}_k[0]$ equals process p_k's input vector.)

Let $\mathbf{v}_{il}[t]$ denote the l-th element of the vector state $\mathbf{v}_i[t]$ of process p_i. Define $\Omega_l[t] = \max_{1 \leq k \leq m} \mathbf{v}_{kl}[t]$, the maximum value of l-th element of the vector state of non-faulty processes. Define $\mu_l[t] = \min_{1 \leq k \leq m} \mathbf{v}_{kl}[t]$, the minimum value of l-th element of the vector state of non-faulty processes. Appendix E proves, using *Observations 1* and *3* above, that

$$\Omega_l[t] - \mu_l[t] \leq (1-\gamma)\left(\Omega_l[t-1] - \mu_l[t-1]\right), \quad \text{for } 1 \leq l \leq d \tag{12}$$

By repeated application of (12) we get

$$\Omega_l[t] - \mu_l[t] \leq (1-\gamma)^t \left(\Omega_l[0] - \mu_l[0]\right) \tag{13}$$

[2] As noted earlier, we omit the round index $[t]$ when discussing the sets $B_i[t]$ and $B_j[t]$ here.

[3] Note that, to simplify the notation somewhat, the notation $g(i,j)$ does not make the round index t explicit. However, it should be noted that $g(i,j)$ for processes p_i and p_j can be different in different rounds.

Therefore, for a given $\epsilon > 0$, if

$$t \;>\; \log_{1/(1-\gamma)} \;\; \frac{\Omega_l[0] - \mu_l[0]}{\epsilon}, \tag{14}$$

then

$$\Omega_l[t] - \mu_l[t] \;<\; \epsilon. \tag{15}$$

Since (14) and (15) hold for $1 \leq l \leq d$, and $U \geq \Omega_l[0]$ and $\nu \leq \mu_l[0]$ for $1 \leq l \leq d$, if each non-faulty process terminates after $1 + \lceil \log_{1/(1-\gamma)} \frac{U-\nu}{\epsilon} \rceil$ rounds, ϵ-agreement is ensured. As shown previously, validity condition is satisfied as well. Thus, the proposed algorithm is correct, and $n \geq (d+2)f+1$ is sufficient for approximate consensus in asynchronous systems. □

4 Simpler Approximate BVC Algorithms with Restricted Round Structure

The proposed approximate BVC algorithm relies on *Component #1* of AAD for exchange of state information among the processes. The communication pattern of AAD requires three message delays in each round (i.e., a causal chain of three messages per round), to ensure strong properties for sets $B_i[t]$, as summarized in Section 3.2. In this section, we consider simpler (restricted) round structure that reduces the communication delay, and the number of messages, per round. The price of the reduction in message cost/delay is an increase in the number of processes necessary to achieve approximate BVC, as seen below.

We consider a restricted round structure for achieving *approximate* consensus in *synchronous* and *asynchronous* settings both. In both settings, each process p_i maintains state $\mathbf{v}_i[t]$, as in the case of the algorithm in Section 3.2. $\mathbf{v}_i[0]$ is initialized to the input vector at process p_i.

Synchronous approximate BVC: The restricted algorithm structure for a synchronous system is as follows. The algorithm executes in synchronous rounds, and each process p_i performs the following steps in the t-th round, $t > 0$.

1. Transmit current vector state, $\mathbf{v}_i[t-1]$, to all the processes. Receive vector state from all the processes. If a message is not received from some process, then its vector state is assumed to have some default value (e.g., the all-0 vector).
2. Compute new state $\mathbf{v}_i[t]$ as a function of $\mathbf{v}_i[t-1]$ and the vectors received from the other processes in the above step.

Asynchronous approximate BVC: The restricted structure of the asynchronous rounds in the asynchronous setting is similar to that in [5]. The messages in this case are tagged by the round index, as in Section 3.2. Each process p_i performs the following steps in its t-th round, $t > 0$:

1. Transmit current state $\mathbf{v}_i[t-1]$ to all the processes. These messages are tagged by round index t.

   Wait until a message tagged by round index t is received from $(n-f-1)$ *other* processes.

2. Compute new state $\mathbf{v}_i[t]$ as a function of $\mathbf{v}_i[t-1]$, and the $(n-f-1)$ other vectors collected in the previous step (for a total of $n-f$ vectors).

For algorithms with the above round structures, the following results can be proved; the proofs are similar to those in Section 3.

Theorem 6 *For the restricted synchronous and asynchronous round structures presented above in Section 4, following conditions are necessary and sufficient:*

- *Synchronous case:* $n \geq (d+2)f+1$
- *Asynchronous case:* $n \geq (d+4)f+1$

To avoid repeating the ideas used in Section 3, we do not present complete formal proofs here. We can prove sufficiency constructively. The restricted round structures above already specify the Step 1 of each round. We can use Step 2 analogous to that of the algorithm in Section 3.2, with $B_i[t]$ being redefined as the set of vectors received by process p_i in Step 1 of the restricted structure.

- In the synchronous setting, $n \geq (d+2)f+1$ is necessary. With $n \geq (d+2)f+1$, observe that any two non-faulty processes p_i and p_j will receive identical vectors from $n-f \geq (d+1)f+1$ non-faulty processes. Thus, $B_i[t] \cap B_j[t]$ contains at least $(d+1)f+1$ identical vectors.
- In the asynchronous setting, $n \geq (d+4)f+1$ is necessary. With $n \geq (d+4)f+1$, each non-faulty processe will have, in Step 2, vectors from at least $n-f$ processes (including itself). Thus, any two fault-free processes will have, in Step 2, vectors from at least $n-2f$ identical processes, of which at most f may be faulty. Thus, $B_i[t] \cap B_j[t]$ contains at least $n-3f$ identical vectors (corresponding to the state of $n-3f$ non-faulty processes). Note that $n-3f \geq (d+1)f+1$.

The proof of correctness of the algorithm in Section 3.2 relies crucially on the property that

$$|B_i[t] \cap B_j[t]| \ \geq \ (d+1)f+1.$$

As discussed above, when the number of nodes satifies the constraints in Theorem 6, this property holds for the restricted round structures too. The rest of the proof of correctness of the restricted algorithms is then similar to the proof of Theorem 4. Thus, the above synchronous and asynchronous algorithms can achieve approximate BVC.

5 Summary

This paper addresses Byzantine vector consensus (BVC) wherein the input at each process, and its decision, is a d-dimensional vector. We derive tight necessary and sufficient bounds on the number of processes required for *Exact BVC* in synchronous systems, and *Approximate BVC* in asynchronous systems.

In Section 4, we derive bounds on the number of processes required for algorithms with restricted round structures to achieve *approximate* consensus in synchronous as well as asynchronous systems.

Acknowledgments

Nitin Vaidya acknowledges Eli Gafni for suggesting the problem of vector consensus, Lewis Tseng for feedback, and Jennifer Welch for answering queries on distributed computing. Vijay Garg acknowledges John Bridgman and Constantine Caramanis for discussions on the problem.

References

[1] I. Abraham, Y. Amit, and D. Dolev. Optimal resilience asynchronous approximate agreement. In *OPODIS*, 2004.

[2] P. Agarwal, M. Sharir, and E. Welzl. Algorithms for center and Tverberg points. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 61–67. ACM, 2004.

[3] M. Ben-Or, D. Dolev, and E. Hoch. Simple gradecast based algorithms. *arXiv preprint arXiv:1007.1049*, 2010.

[4] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[5] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33:499–516, May 1986.

[6] D. Dolev, R. Reischuk, and H. Strong. Early stopping in byzantine agreement. *Journal of the ACM (JACM)*, 37(4):720–741, 1990.

[7] A. Doudou and A. Schiper. Muteness detector for consensus with Byzantine processes. In *ACM PODC*, 1998.

[8] A. D. Fekete. Asymptotically optimal algorithms for approximate agreement. In *Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, PODC '86, pages 73–87, New York, NY, USA, 1986. ACM.

[9] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32:374–382, April 1985.

[10] J. Garay and Y. Moses. Fully polynomial byzantine agreement for processors in rounds. *SIAM Journal on Computing*, 27(1):247–290, 1998.

[11] S. Jadhav and A. Mukhopadhyay. Computing a centerpoint of a finite planar set of points in linear time. *Discrete & Computational Geometry*, 1994.

[12] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. Prog. Lang. Syst.*, 4(3):382–401, July 1982.

[13] N. A. Lynch. *Distributed algorithms*. Morgan Kaufmann Publishers, 1995.

[14] G. Miller and D. Sheehy. Approximate centerpoints with proofs. *Computational Geometry*, 43(8):647–654, 2010.

[15] W. Mulzer and D. Werner. Approximating Tverberg points in linear time for any fixed dimension. In *Proceedings of the 2012 symposuim on Computational Geometry*, pages 303–310. ACM, 2012.

[16] N. Neves, M. Correia, and P. Verissimo. Solving vector consensus with a wormhole. *IEEE Trans. on Parallel and Distributed Systems*, December 2005.

[17] M. A. Perles and M. Sigron. A generalization of Tverberg's theorem, 2007. CoRR, http://arxiv.org/abs/0710.4668.

[18] N. H. Vaidya, L. Tseng, and G. Liang. Iterative approximate byzantine consensus in arbitrary directed graphs. In *ACM Symposium on Principles of Distributed Computing (PODC)*, July 2012.

Appendix

A Notations

This appendix summarizes some of the notations and terminology introduced in the paper.

- n = number of processes.
- $\mathcal{P} = \{p_1, p_2, \cdots, p_n\}$ is the set of processes in the system.
- f = maximum number of Byzantine faulty processes.
- d = dimension of the input vector as well as decision vector at each process.
- $\mathbf{x}_i$ = d-dimensional input vector at process p_i. The vector is equivalently viewed as a point in the Euclidean space $\mathbf{R}^d$.
- $\mathcal{H}(Y)$ denotes the convex hull of the points in multiset Y.
- m : The proof of Theorem 5 assumes, without loss of generality, that for some $m \geq n - f$, processes $p_1, \cdots, p_m$ are non-faulty, and the remaining $n - m$ processes are faulty.
- $\Gamma(.)$ is defined in (1).
- $\Phi(.)$ is defined in (8).
- $\mathbf{v}_i[t]$ is the state of process p_i at the end of its t-th round of the asynchronous BVC algorithm, $t > 0$. Thus, $\mathbf{v}_i[t-1]$ is the state of process p_i at the start of its t-th round, $t > 0$. $\mathbf{v}_i[0]$ for process p_i equals its input $\mathbf{x}_i$.
- $\mathbf{v}_{il}[t]$ is the l-th element of $\mathbf{v}_i[t]$, where $1 \leq l \leq d$.
- $B_i[t]$ defined in Section 3.2, is a set of tuples of the form $(p_j, \mathbf{w}_j, t)$, obtained by process p_i in Step 1 of the approximate consensus algorithm.
- *Weight* in a convex combination is defined in Definition 1
- $\gamma = \frac{1}{n\binom{n}{n-f}}$, as defined in (11). Note that $0 < \gamma < 1$ for finite $n > 1$.
- $\Omega_l[t] = \max_{1 \leq k \leq m} \ \mathbf{v}_{kl}[t]$
- $\mu_l[t] = \min_{1 \leq k \leq m} \ \mathbf{v}_{kl}[t]$
- $\rho_l[t] = \Omega_l[t] - \mu_l[t]$
- $|Y|$ denotes the size of a *multiset* Y.
- $\| \ a \ \|$ is the absolute value of a real number a.

B Multisets and Multiset Partition

Multiset is a generalization on the notion of a set. While the members in a set must be distinct, a multiset may contain the same member multiple times.

Notions of a *subset of a multiset* and a *partition of a multiset* have natural definitions. For completeness, we present the definitions here.

Suppose that Y is a multiset. Y contains $|Y|$ members. Denote the members in Y as y_i, $1 \leq i \leq |Y|$. Thus, $Y = \{y_1, y_2, \cdots, y_{|Y|}\}$. Define set $N_Y = \{1, 2, \cdots, |Y|\}$. Thus, N_Y contains integers from 1 to $|Y|$. Since Y is a multiset, it is possible that $y_i = y_j$ for some $i \neq j$.

Z is a subset of Y provided that there exists a set $N_Z \subseteq N_Y$ such that

$$Z = \{y_i \ : \ i \in N_Z\}$$

Subsets $Y_1, Y_2, \cdots, Y_b$ of multiset Y form a partition of Y provided that there exists a partition $N_1, N_2, \cdots, N_b$ of set N_Y such that

$$Y_j = \{y_i \ : \ i \in N_j\}, \quad 1 \leq j \leq b$$

C Clarification for the Proof of Theorem 1

In the proof of Theorem 1, when considering the case of $f = 1$, we claimed the following:

> Since none of the non-faulty processes know which process, if any, is faulty, as elaborated in Appendix C, the decision vector must be in the convex hull of each multiset containing the input vectors of $n - 1$ of the processes (there are n such multisets). Thus, this intersection must be non-empty, for all possible input vectors at the n processes.

Now we provide an explanation for the above claim.

Suppose that the input at process p_i is $\mathbf{x}_i$, $1 \leq i \leq n$. All the processes are non-faulty, but the processes do not know this fact. The decision vector chosen by the processes must satisfy the *agreement* and *validity* conditions both.

- With $f = 1$, any one process may potentially be faulty. In particular, process p_i ($1 \leq i \leq n$) may possibly be faulty. Therefore, the input $\mathbf{x}_i$ of process p_i cannot be trusted by other processes. Then to ensure *validity*, the decision vector chosen by any other process p_j ($j \neq i$) must be in the convex hull of the inputs at the processes in $\mathcal{P} - \{p_i\}$ (i.e., all processes except p_i). Thus, the decision vector of process p_j ($j \neq i$) must be in the convex hull of the points in multiset X^i below.
$$X^i = \{\mathbf{x}_k \ : \ k \neq i, \ 1 \leq k \leq n\}.$$
- To ensure *agreement*, the decision vector chosen by all the processes must be identical. Therefore, the decision vector must be in the intersection of the convex hulls of all the multisets X^i ($1 \leq i \leq n$) defined above. Thus, we conclude that the decision vector must be in the intersection below, where $\mathcal{H}(X^i)$ denotes the *convex hull* of the points in multiset X^i, and Q_i denotes $\mathcal{H}(X^i)$.
$$\cap_{i=1}^{n} \ \mathcal{H}(X^i) \ = \ \cap_{i=1}^{n} \ Q_i \tag{16}$$

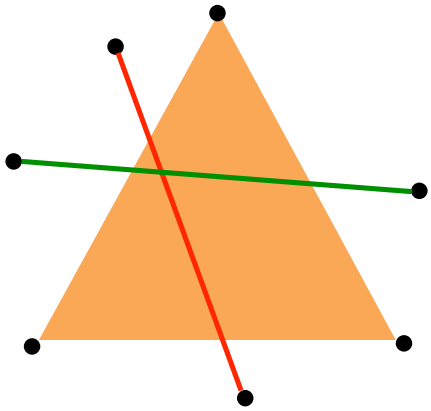

Figure 1: *Illustration of a Tverberg partition.*
Acknowledgment*: The above example is inspired by an illustration authored by David Eppstein, which is available in the public domain from* Wikipedia Commons.

If the intersection in (16) is empty, then there is no decision vector that satisfies *validity* and *agreement* conditions both. Therefore, the intersection must be non-empty.

As shown in the proof of Theorem 1, if n is not large enough, then the intersection in (16) may be empty.

D Tverberg Partition

Figure 1 illustrates a Tverberg partition of a set of 7 vertices in 2-dimensions. The 7 vertices are at the corners of a heptagon. Thus, $n = 7$ here, and $d = 2$. Let $f = 2$. Then, $n = (d+1)f + 1$, and Tverberg's Theorem 2 implies the presence of a Tverberg partition consisting of $f + 1 = 3$ subsets. Figure 1 shows the convex hulls of the three subsets in the Tverberg partition: one convex hull is a triangle, and the other two convex hulls are each a line segment. In this example, the three convex hulls intersect in exactly one point. Thus, there is just one Tverberg point. In general, there can be multiple Tverberg points.

E Proof of (12)

$\mathbf{v}_{il}[t]$ denotes the l-th element of the vector state $\mathbf{v}_i[t]$ of process p_i, $1 \leq l \leq d$. Processes $p_1, \cdots, p_m$ are non-faulty, and processes $p_{m+1}, \cdots, p_n$ are faulty, where $m \geq n - f$. Recall that, for $1 \leq l \leq d$,

$$\Omega_l[t] = \max_{1 \leq k \leq m} \mathbf{v}_{kl}[t], \text{ maximum value of } l\text{-th elements at non-faulty processes} \quad (17)$$

$$\mu_l[t] = \min_{1 \leq k \leq m} \mathbf{v}_{kl}[t], \text{ minimum value of } l\text{-th elements at non-faulty processes} \quad (18)$$

Define (19)

$$\rho_l[t] = \Omega_l[t] - \mu_l[t] \quad (20)$$

Equivalently,

$$\rho_l[t] = \max_{1 \leq i,j \leq m} \| \mathbf{v}_{il}[t] - \mathbf{v}_{jl}[t] \| \quad (21)$$

where $\| \, . \, \|$ operator yields the absolute value of the scalar parameter.

Consider any two non-faulty processes p_i, p_j (thus, $1 \leq i, j \leq m$). Consider $1 \leq l \leq d$. Then

$$\mu_l[t-1] \;\leq\; \mathbf{v}_{il}[t-1] \;\leq\; \Omega_l[t-1] \tag{22}$$

$$\mu_l[t-1] \;\leq\; \mathbf{v}_{jl}[t-1] \;\leq\; \Omega_l[t-1] \tag{23}$$

Observations 1 and *3* in Part III of the proof of Theorem 5, and the definition of γ, imply the existence of constants α_k's and β_k's such that:

$$\mathbf{v}_i[t] \;=\; \sum_{k=1}^{m} \alpha_k \, \mathbf{v}_k[t-1] \quad \text{where} \tag{24}$$

$$\alpha_k \geq 0 \text{ for } 1 \leq k \leq m, \;\; \text{and} \;\; \sum_{k=1}^{m} \alpha_k = 1 \tag{25}$$

$$\alpha_{g(i,j)} \geq \gamma \tag{26}$$

$$\mathbf{v}_j[t] \;=\; \sum_{k=1}^{m} \beta_k \, \mathbf{v}_k[t-1] \quad \text{where} \tag{27}$$

$$\beta_k \geq 0 \text{ for } 1 \leq k \leq m, \;\; \text{and} \;\; \sum_{k=1}^{m} \beta_k = 1 \tag{28}$$

$$\beta_{g(i,j)} \geq \gamma \tag{29}$$

In the following, let us abbreviate $g(i,j)$ simply as g. Thus, $\alpha_{g(i,j)}$ is same as α_g, and $\beta_{g(i,j)}$ is same as β_g. From (24) and (27), focussing on just the operations on l-th elements, we obtain

$$\begin{aligned}
\mathbf{v}_{il}[t] \;&=\; \sum_{k=1}^{m} \alpha_k \, \mathbf{v}_{kl}[t-1] \\
&\leq\; \alpha_g \, \mathbf{v}_{gl}[t-1] \;+\; (1-\alpha_g) \, \Omega_l[t-1] \quad \text{because } \mathbf{v}_{kl}[t-1] \leq \Omega_l[t-1], \; \forall k \\
&\leq\; \gamma \, \mathbf{v}_{gl}[t-1] \;+\; (\alpha_g - \gamma) \mathbf{v}_{gl}[t-1] \;+\; (1-\alpha_g) \, \Omega_l[t-1] \\
&\leq\; \gamma \, \mathbf{v}_{gl}[t-1] \;+\; (\alpha_g - \gamma) \Omega_l[t-1] \;+\; (1-\alpha_g) \, \Omega_l[t-1] \\
&\qquad\qquad \text{because } \mathbf{v}_{gl}[t-1] \leq \Omega_l[t-1] \text{ and } \alpha_g \geq \gamma \\
&\leq\; \gamma \, \mathbf{v}_{gl}[t-1] \;+\; (1-\gamma) \, \Omega_l[t-1]
\end{aligned} \tag{30}$$

$$\begin{aligned}
\mathbf{v}_{jl}[t] \;&=\; \sum_{k=1}^{m} \beta_k \, \mathbf{v}_{kl}[t-1] \\
&\geq\; \beta_g \, \mathbf{v}_{gl}[t-1] \;+\; (1-\beta_g) \, \mu_l[t-1] \quad \text{because } \mathbf{v}_{kl}[t-1] \geq \mu_l[t-1], \; \forall k \\
&\geq\; \gamma \, \mathbf{v}_{gl}[t-1] \;+\; (\beta_g - \gamma) \mathbf{v}_{gl}[t-1] \;+\; (1-\beta_g) \, \mu_l[t-1] \\
&\geq\; \gamma \, \mathbf{v}_{gl}[t-1] \;+\; (\beta_g - \gamma) \mu_l[t-1] \;+\; (1-\beta_g) \, \mu_l[t-1] \\
&\qquad\qquad \text{because } \mathbf{v}_{gl}[t-1] \geq \mu_l[t-1], \text{ and } \beta_g \geq \gamma \\
&\geq\; \gamma \, \mathbf{v}_{gl}[t-1] \;+\; (1-\gamma) \, \mu_l[t-1]
\end{aligned} \tag{31}$$

$$\Rightarrow \quad \mathbf{v}_{il}[t] \;-\; \mathbf{v}_{jl}[t] \;\leq\; (1-\gamma) \, (\Omega_l[t-1] - \mu_l[t-1]) \quad \text{subtracting (31) from (30)} \tag{32}$$

By swapping the role of p_i and p_j above, we can also show that

$$\mathbf{v}_{jl}[t] \ - \ \mathbf{v}_{il}[t] \ \leq \ (1-\gamma)\,(\Omega_l[t-1] - \mu_l[t-1]) \tag{33}$$

Putting (32) and (33) together, we obtain

$$\begin{aligned} \| \ \mathbf{v}_{il}[t] \ - \ \mathbf{v}_{jl}[t] \ \| \ &\leq \ (1-\gamma)\,(\Omega_l[t-1] - \mu_l[t-1]) \ \text{ because } \Omega_l[t-1] \geq \mu_l[t-1] \\ &\leq \ (1-\gamma)\,\rho_l[t-1] \quad \text{by the definition of } \rho_l[t-1] \end{aligned} \tag{34}$$

$$\Rightarrow \quad \max_{1 \leq i,j \leq m} \ \| \ \mathbf{v}_{il}[t] \ - \ \mathbf{v}_{jl}[t] \ \| \ \leq \ (1-\gamma)\,\rho_l[t-1] \tag{35}$$

because the previous inequality holds for all $1 \leq i, j \leq m$

$$\Rightarrow \quad \rho_l[t] \ \leq \ (1-\gamma)\,\rho_l[t-1] \quad \text{by (21)} \tag{36}$$

$$\Rightarrow \quad \Omega_l[t] - \mu_l[t] \ \leq \ (1-\gamma)\,(\Omega_l[t-1] - \mu_l[t-1]) \quad \text{by definition of } \rho_l[t]$$

This proves (12).

F Optimization of Step 2 of Asynchronous BVC

Property 1 of *Component #1* of AAD described in Section 3.2 is a consequence of a stronger property satisfied by the AAD algorithm.

In AAD, each process p_k sends out notifications to others each time it adds a new tuple to its $B_k[t]$; the notifications are sent over the FIFO links. AAD defines a process p_k to be a "witness" for process p_i provided that (i) p_k is known to have added at least $n-f$ tuples to $B_k[t]$, and (ii) all the tuples that p_k claims to have added to $B_k[t]$ are also in $B_i[t]$.

AAD also ensures that each non-faulty process has at least $n-f$ witnesses, ensuring that any two non-faulty processes have at least $n-2f$ witnesses in common, where $n-2f \geq f+1$. Thus, any two non-faulty processes p_i and p_j have at least one non-faulty witness in common, say p_k. This, in turn, ensures (due to the manner in which the advertisements above are sent) that $B_i[t] \cap B_j[t]$ contains at least the first $n-f$ tuples advertised by p_k.

Each process can keep track of the order in which the tuples advertised by each process are received. Then, in Step 2 of the asynchronous approximate BVC algorithm, instead of enumerating all the $n-f$-size subsets C of $B_i[t]$, it suffices to only consider those subsets of $B_i[t]$ that correspond to the first $n-f$ tuples advertised by each witness of p_i. Since there can be no more than n witnesses, at most n sets C need to be considered. Thus, in this case $|Z_i| \leq n$.

Since each pair of non-faulty processes p_i and p_j shares a non-faulty witness, despite considering only $\leq n$ subsets in Step 2, Z_i and Z_j computed by p_i and p_j contain at least one identical point, say, $\mathbf{z}_{ij}$. Our proof of correctness of the algorithm relied on the existence of such a point.

It should now be easy to see that the rest of the proof of correctness will remain the same, with γ being re-defined as

$$\gamma \ = \ \frac{1}{n^2}.$$